Bidirectional Context Sensitive data flow analysis in PRISM

Vinit Deodhar
(*Poject Guide: Prof. Uday Khedker*)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

June 2014

Outline of the talk

- Overview of PRISM data flow analyzer generator
- Background on Value context method.
- Our extensions to PRISM
- Building a scalable data flow solver
- Performance measurements
- Conclusion and future work

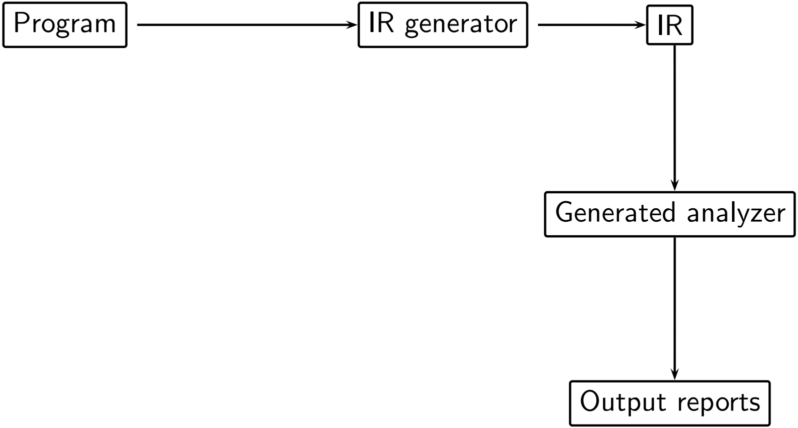
Part I

Overview of PRISM data flow analyzer generator

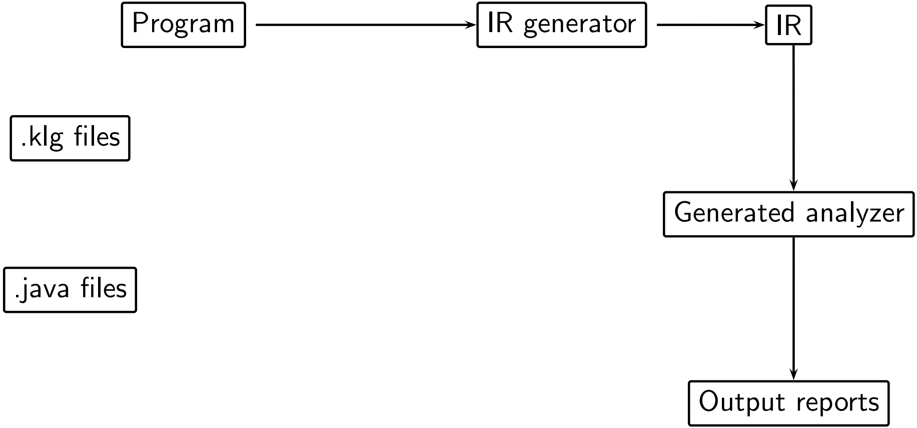
PRISM

- PRISM is an program analyzer generator developed by TATA Research Development and Design Center (TRDDC)

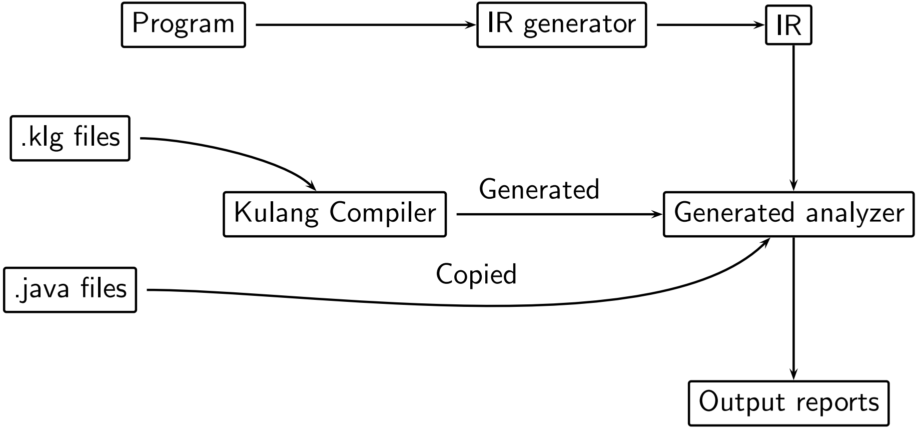
Architecture of PRISM



Architecture of PRISM



Architecture of PRISM



Kulang specifications

```
lattice L ::  set NamedEntity;

top : (set NamedEntity){};

A meet B : A+B;

BoundaryValue : (set NamedEntity){};

BackwardNodeflow( n: Call, S: L )
let
   useincall = getNEsFromCall( n )
in
   S + useincall;
```

Limitations of current version of PRISM

- PRISM solver is not context sensitive

Limitations of current version of PRISM

- PRISM solver is not context sensitive
- The current framework does not support context sensitive analysis. Summary based approach is used in the current solver.

Limitations of current version of PRISM

- PRISM solver is not context sensitive
- The current framework does not support context sensitive analysis. Summary based approach is used in the current solver.
- The current framework does not support bidirectional analysis. Analysis can be either forward of backward. Hence there is no implementation of context sensitive bidirectional analysis.

Limitations of current version of PRISM

- PRISM solver is not context sensitive
- The current framework does not support context sensitive analysis. Summary based approach is used in the current solver.
- The current framework does not support bidirectional analysis. Analysis can be either forward of backward. Hence there is no implementation of context sensitive bidirectional analysis.
- The current kulang specifications does not have support for specifying lattices of data flow problems.

Limitations of current version of PRISM

- PRISM solver is not context sensitive
- The current framework does not support context sensitive analysis. Summary based approach is used in the current solver.
- The current framework does not support bidirectional analysis. Analysis can be either forward of backward. Hence there is no implementation of context sensitive bidirectional analysis.
- The current kulang specifications does not have support for specifying lattices of data flow problems.
- Meet function needs to be explicitly defined in kulang specifications. The meet function can be inferred from the lattice of the data flow problem.

Limitations of current version of PRISM

- PRISM solver is not context sensitive
- The current framework does not support context sensitive analysis. Summary based approach is used in the current solver.
- The current framework does not support bidirectional analysis. Analysis can be either forward of backward. Hence there is no implementation of context sensitive bidirectional analysis.
- The current kulang specifications does not have support for specifying lattices of data flow problems.
- Meet function needs to be explicitly defined in kulang specifications. The meet function can be inferred from the lattice of the data flow problem.
- There is no proper way to debug the kulang specifications.

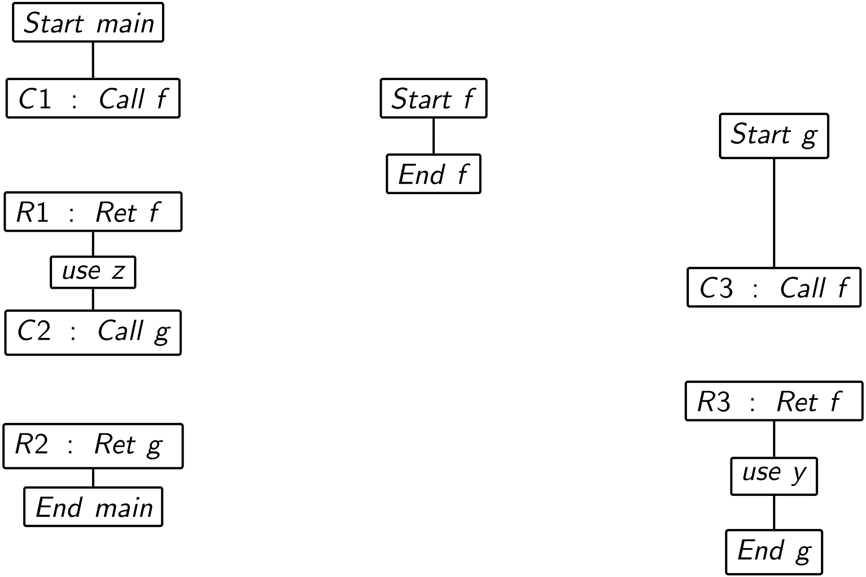
Part II

Value Context method of data flow analysis

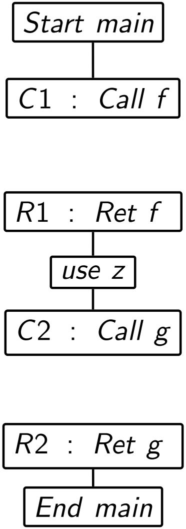
Context Sensitive analysis using value contexts

- Based on paper Interprocedural data flow analysis in Soot using value contexts by Padhye, Rohan and Khedker, Uday presented in SOAP 13
- New context created for a unique (function, entry value) combination
- Uses a context transition graph for expressing relation between contexts

Motivating example of value contexts method

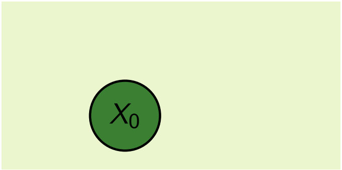


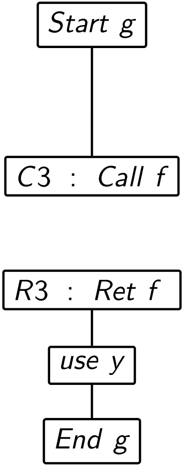
Motivating example of value contexts method



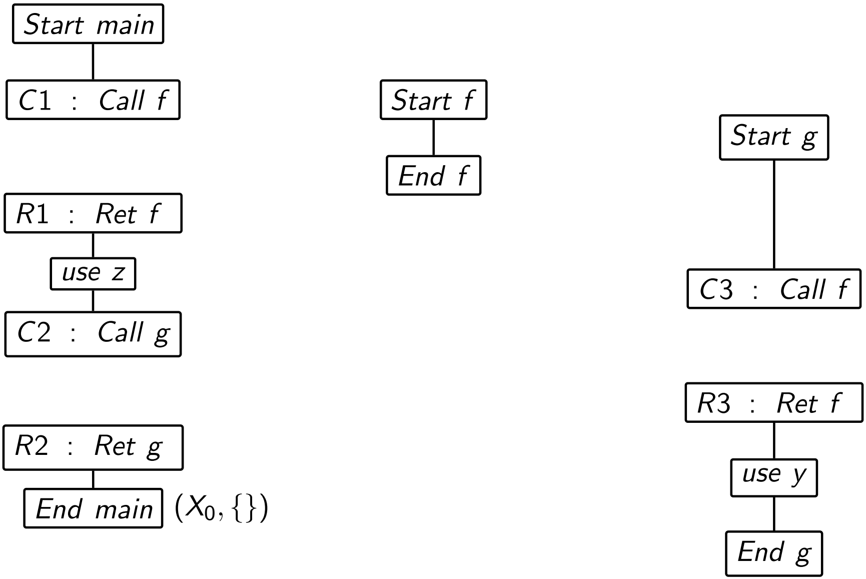
Context table

Context	Function	Entry	Exit
X_0	main	{}	{}

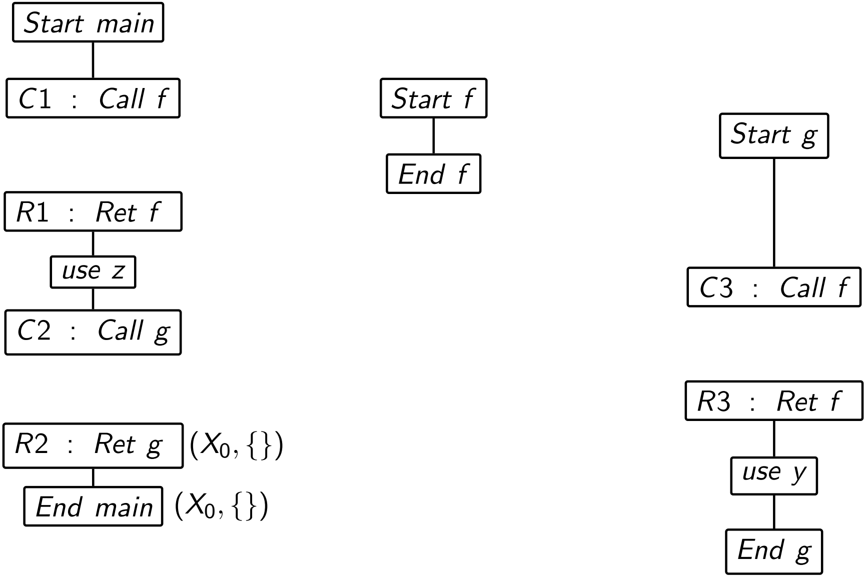




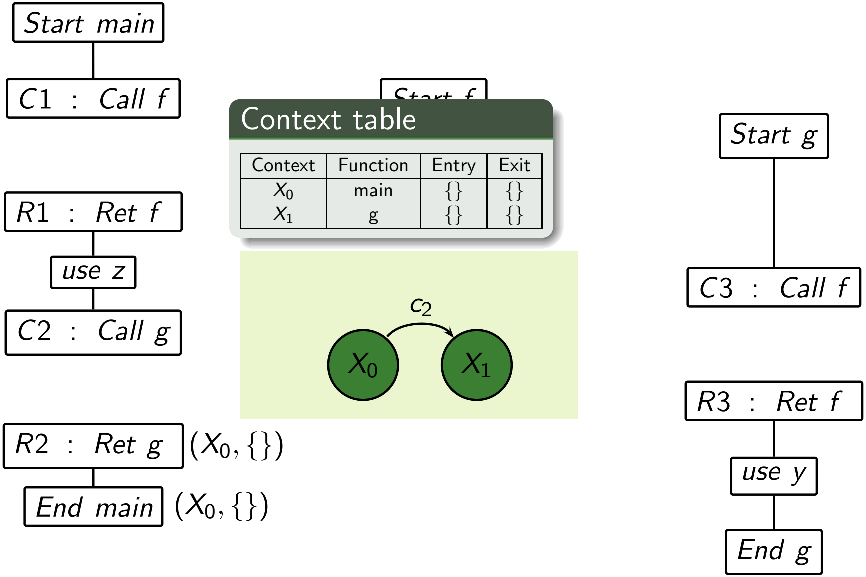
Motivating example of value contexts method



Motivating example of value contexts method



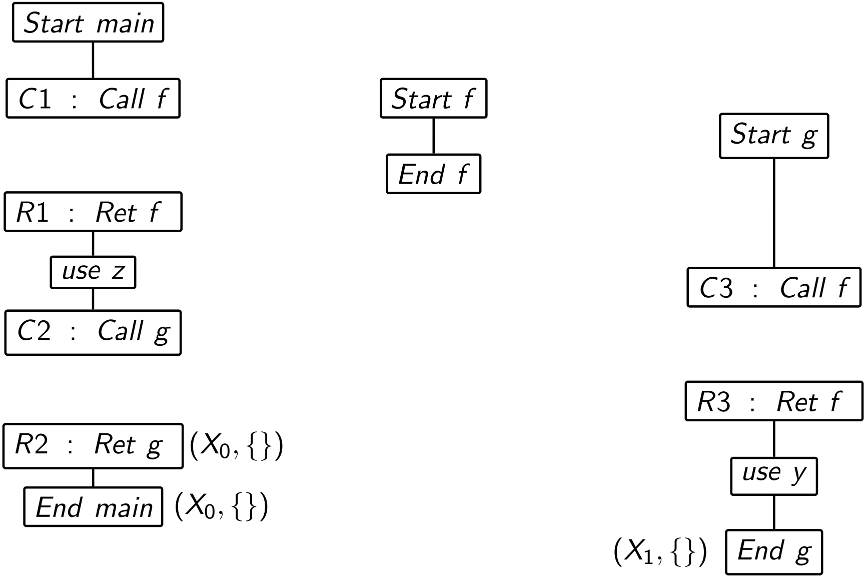
Motivating example of value contexts method



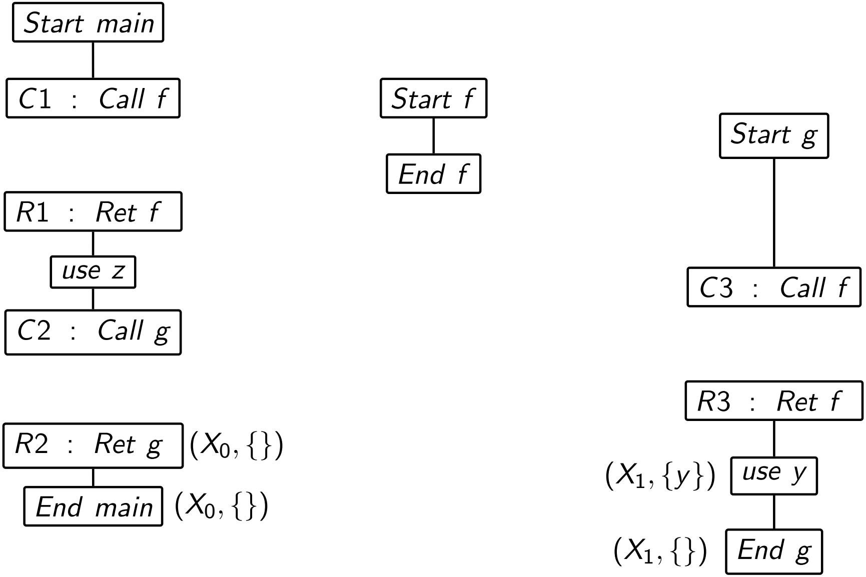
Context table

Context	Function	Entry	Exit
X_0	main	$\{\}$	$\{\}$
X_1	g	$\{\}$	$\{\}$

Motivating example of value contexts method



Motivating example of value contexts method



Start main

*C*1 : *Call f*

*R*1 : *Ret f*

use z

*C*2 : *Call g*

*R*2 : *Ret g* $(X_0, \{\})$

End main $(X_0, \{\})$

Start f

End f

Start g

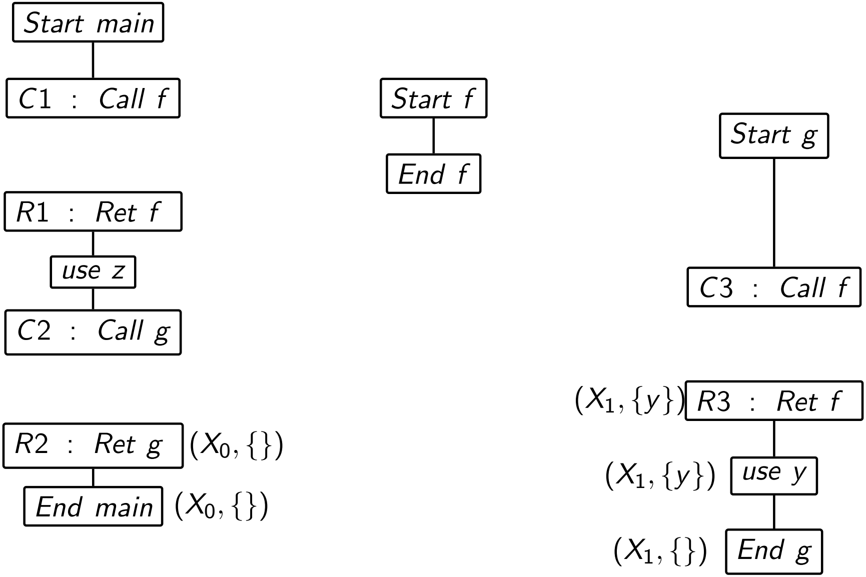
*C*3 : *Call f*

*R*3 : *Ret f*

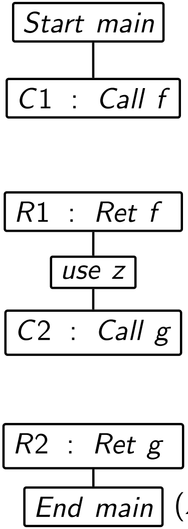
$(X_1, \{y\})$ *use y*

$(X_1, \{\})$ *End g*

Motivating example of value contexts method



Motivating example of value contexts method

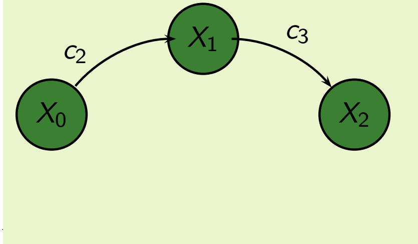


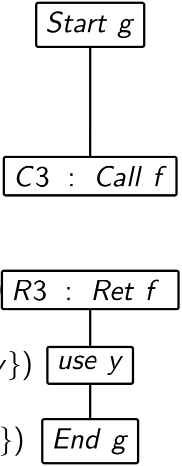
(

$(X_0, \{\})$

Context table

Context	Function	Entry	Exit
X_0	main	{}	{}
X_1	g	{}	{}
X_2	f	{y}	{}



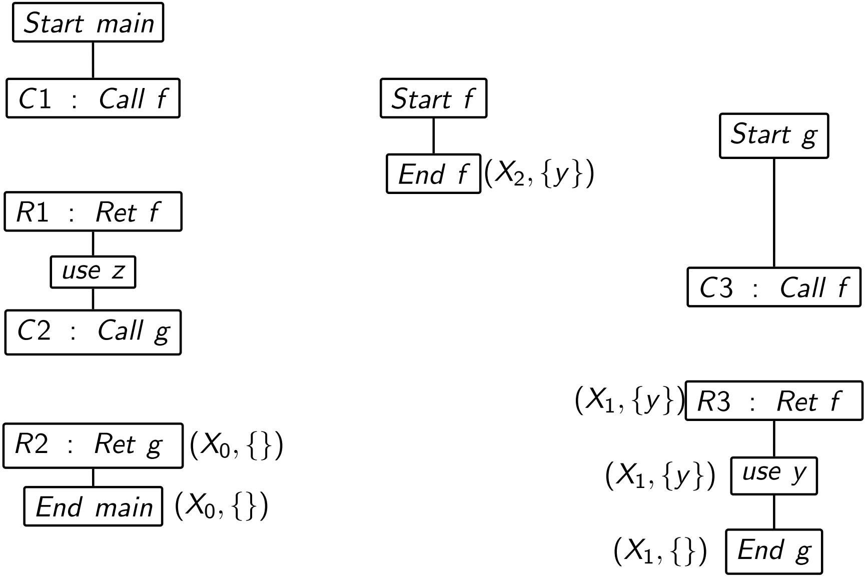


$, \{y\})$

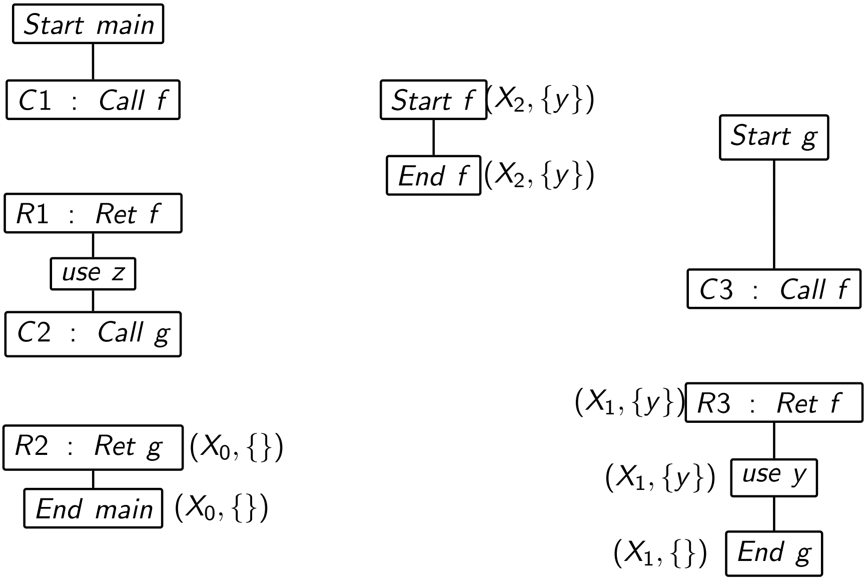
$(X_1, \{y\})$

$(X_1, \{\})$

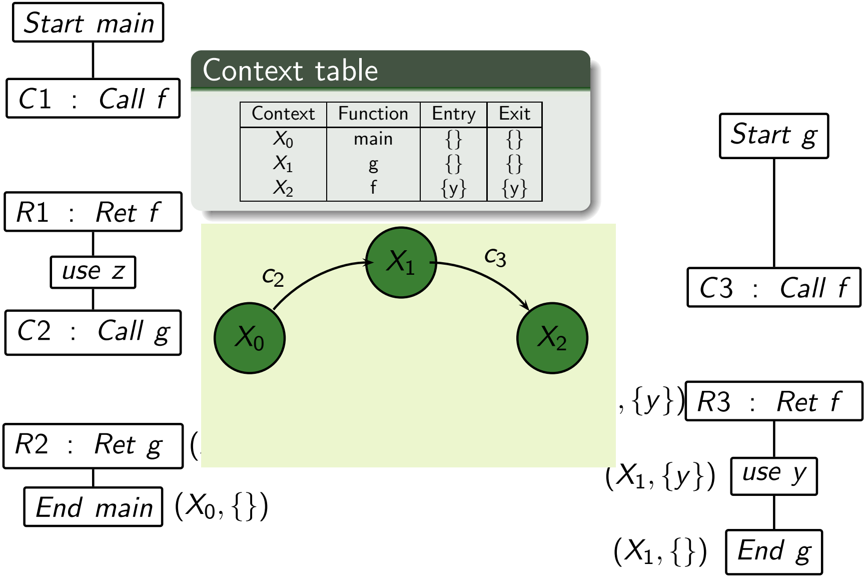
Motivating example of value contexts method



Motivating example of value contexts method



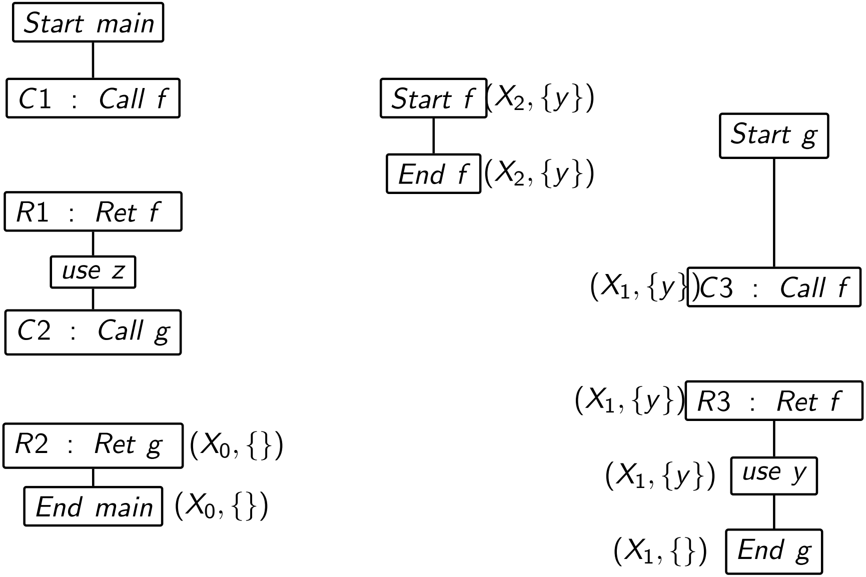
Motivating example of value contexts method



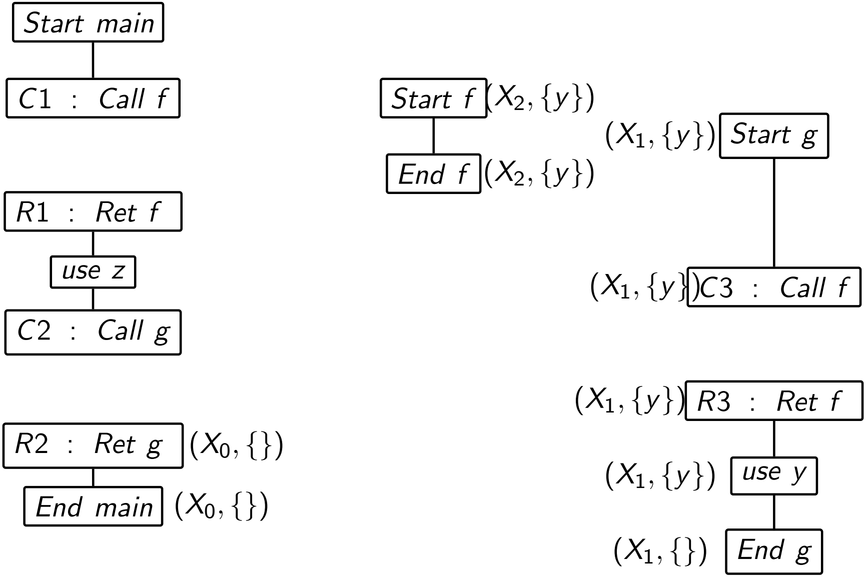
Context table

Context	Function	Entry	Exit
X_0	main	{}	{}
X_1	g	{}	{}
X_2	f	{y}	{y}

Motivating example of value contexts method



Motivating example of value contexts method



Start main

*C*1 : *Call f*

*R*1 : *Ret f*

use z

*C*2 : *Call g*

*R*2 : *Ret g* $(X_0, \{\})$

End main $(X_0, \{\})$

Start f $(X_2, \{y\})$

End f $(X_2, \{y\})$

$(X_1, \{y\})$ *Start g*

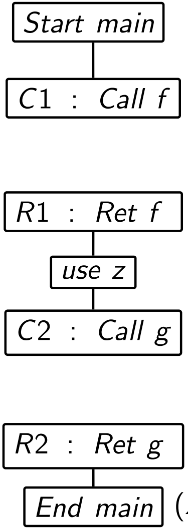
$(X_1, \{y\})$ *C*3 : *Call f*

$(X_1, \{y\})$ *R*3 : *Ret f*

$(X_1, \{y\})$ *use y*

$(X_1, \{\})$ *End g*

Motivating example of value contexts method



Start main

C1 : Call f

R1 : Ret f

use z

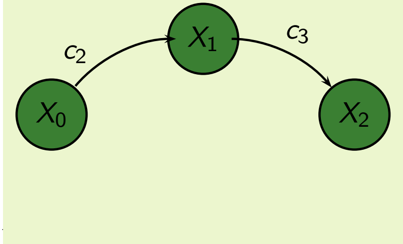
C2 : Call g

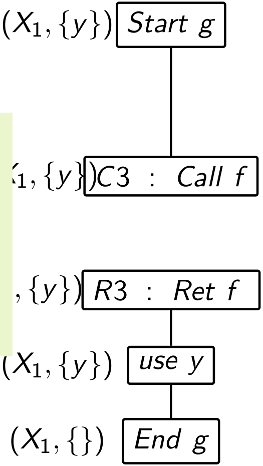
R2 : Ret g (

End main $(X_0, \{\})$

Context table

Context	Function	Entry	Exit
X_0	main	{}	{}
X_1	g	{}	{y}
X_2	f	{y}	{y}





$(X_1, \{y\})$ *Start g*

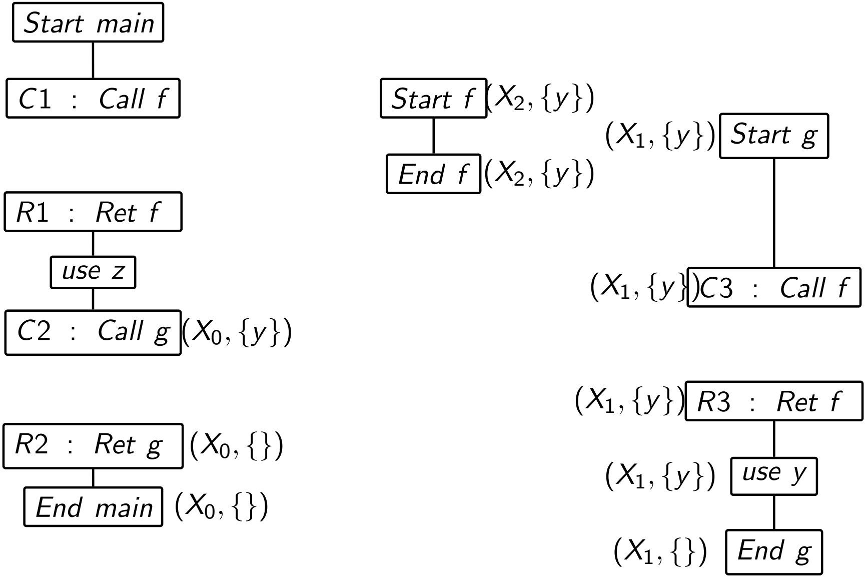
$\!_1, \{y\})$ *C3 : Call f*

$, \{y\})$ *R3 : Ret f*

$(X_1, \{y\})$ *use y*

$(X_1, \{\})$ *End g*

Motivating example of value contexts method



Start main

C1 : Call f

R1 : Ret f

use z

C2 : Call g $(X_0, \{y\})$

R2 : Ret g $(X_0, \{\})$

End main $(X_0, \{\})$

Start f $(X_2, \{y\})$

End f $(X_2, \{y\})$

$(X_1, \{y\})$ *Start g*

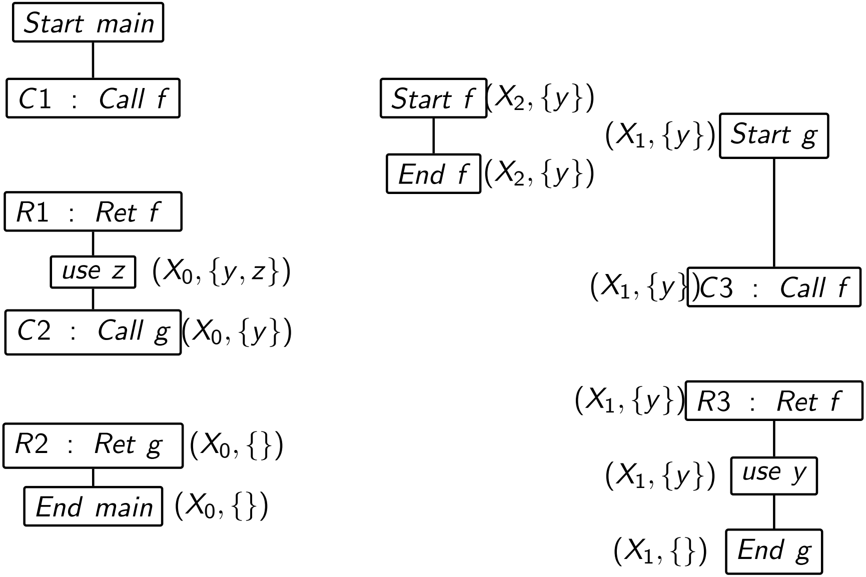
$(X_1, \{y\})$ *C3 : Call f*

$(X_1, \{y\})$ *R3 : Ret f*

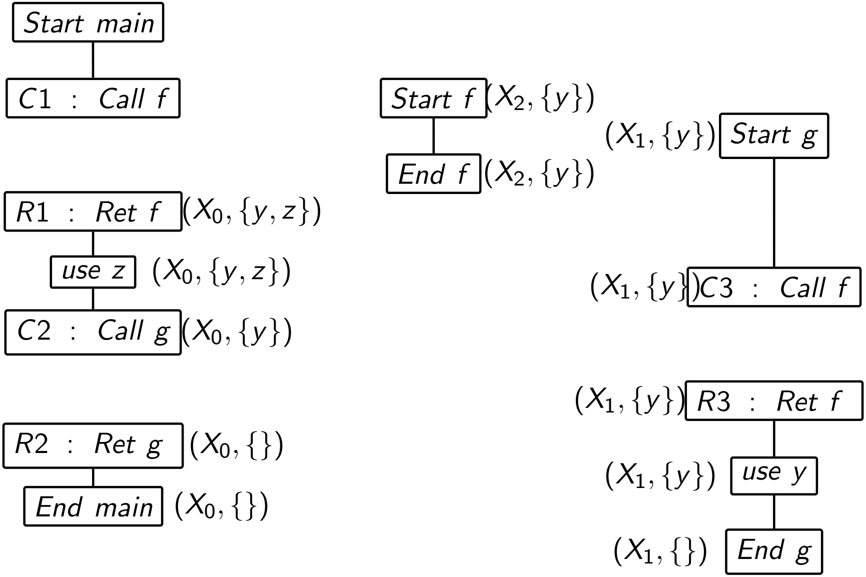
$(X_1, \{y\})$ *use y*

$(X_1, \{\})$ *End g*

Motivating example of value contexts method



Motivating example of value contexts method



$Start\ main$

$C1\ :\ Call\ f$

$R1\ :\ Ret\ f$ $(X_0, \{y, z\})$

$use\ z$ $(X_0, \{y, z\})$

$C2\ :\ Call\ g$ $(X_0, \{y\})$

$R2\ :\ Ret\ g$ $(X_0, \{\})$

$End\ main$ $(X_0, \{\})$

$Start\ f$ $(X_2, \{y\})$

$End\ f$ $(X_2, \{y\})$

$(X_1, \{y\})$ $Start\ g$

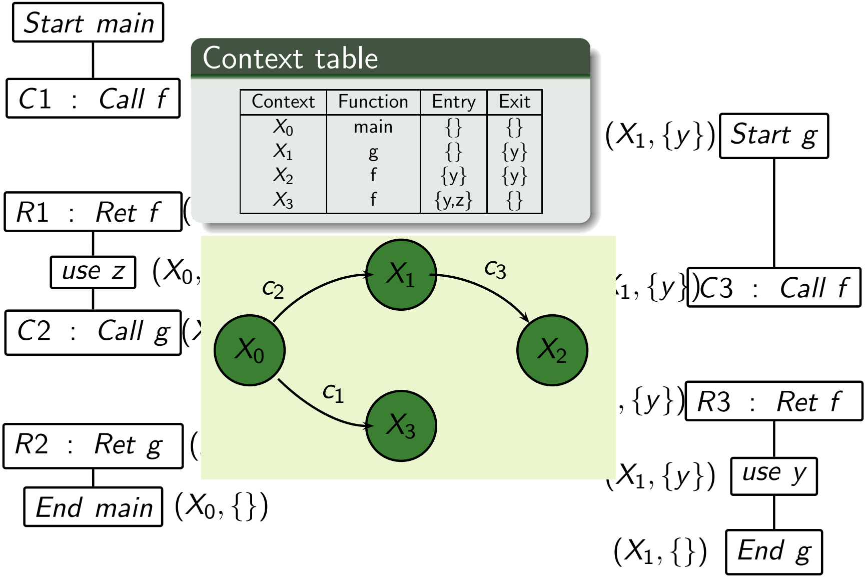
$(X_1, \{y\})$ $C3\ :\ Call\ f$

$(X_1, \{y\})$ $R3\ :\ Ret\ f$

$(X_1, \{y\})$ $use\ y$

$(X_1, \{\})$ $End\ g$

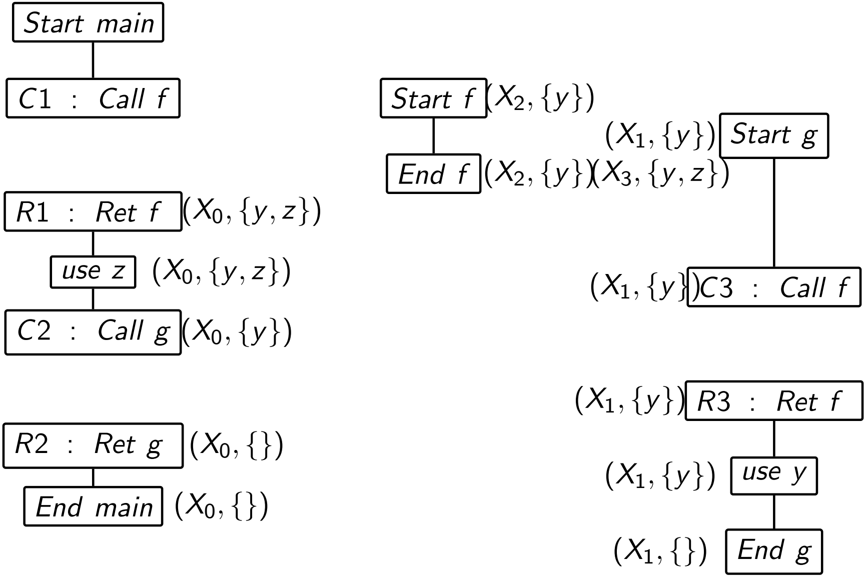
Motivating example of value contexts method



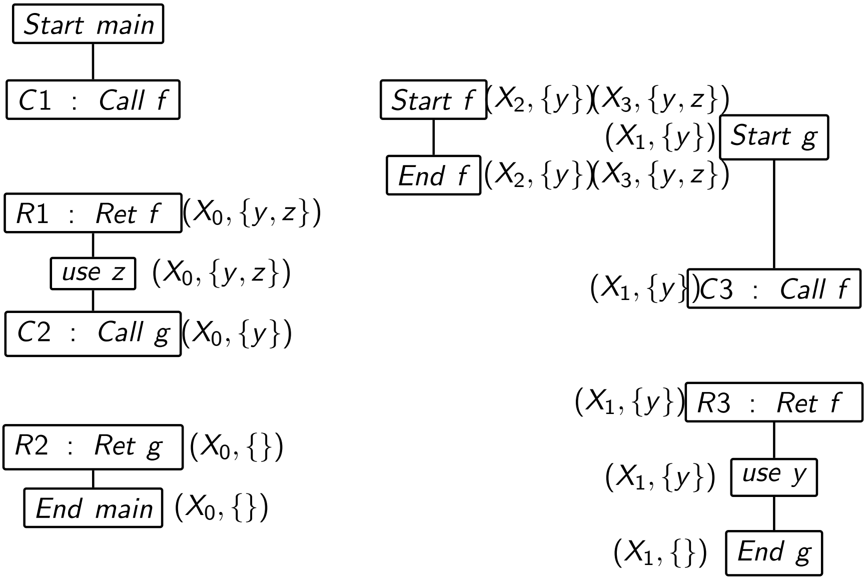
Context table

Context	Function	Entry	Exit
X_0	main	{}	{}
X_1	g	{}	{y}
X_2	f	{y}	{y}
X_3	f	{y,z}	{}

Motivating example of value contexts method



Motivating example of value contexts method



Motivating example of value contexts method

Start main

C1 : Call f

R1 : Ret f (

use z $(X_0,$

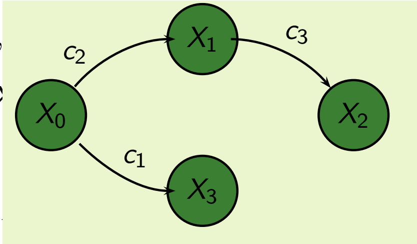
C2 : Call g $(X$

R2 : Ret g (

End main $(X_0, \{\})$

Context table

Context	Function	Entry	Exit
X_0	main	{}	{}
X_1	g	{}	{y}
X_2	f	{y}	{y}
X_3	f	{y,z}	{y,z}



$X_3, \{y, z\})$

$(X_1, \{y\})$ *Start g*

$X_3, \{y, z\})$

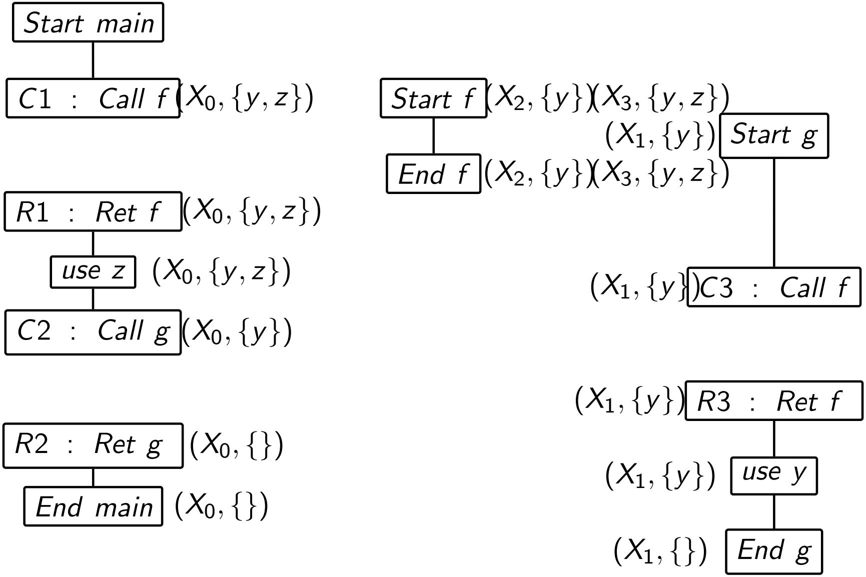
$X_1, \{y\})$ *C3 : Call f*

$, \{y\})$ *R3 : Ret f*

$(X_1, \{y\})$ *use y*

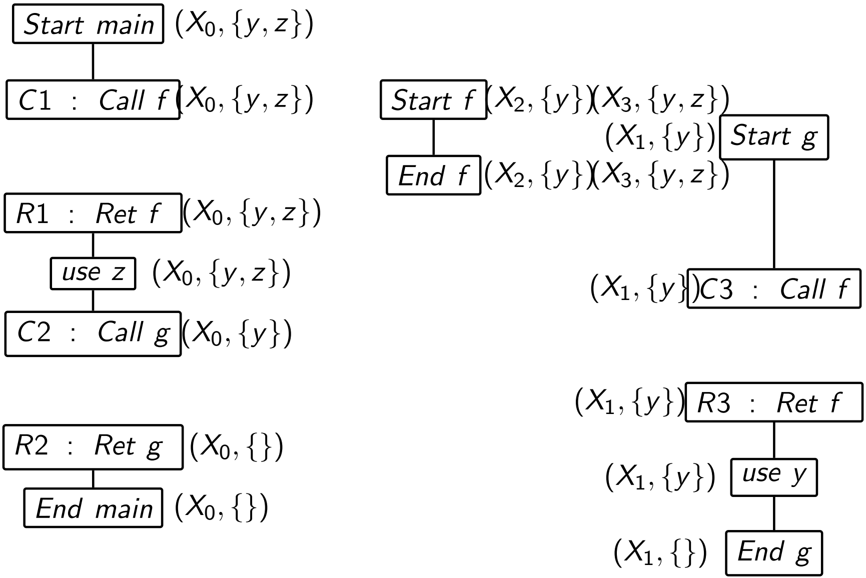
$(X_1, \{\})$ *End g*

Motivating example of value contexts method



Start main
*C*1 : *Call f* $(X_0, \{y, z\})$
*R*1 : *Ret f* $(X_0, \{y, z\})$
use z $(X_0, \{y, z\})$
*C*2 : *Call g* $(X_0, \{y\})$
*R*2 : *Ret g* $(X_0, \{\})$
End main $(X_0, \{\})$
Start f $(X_2, \{y\})(X_3, \{y, z\})$
End f $(X_2, \{y\})(X_3, \{y, z\})$
$(X_1, \{y\})$ *Start g*
$(X_1, \{y\})$ *C*3 : *Call f*
$(X_1, \{y\})$ *R*3 : *Ret f*
$(X_1, \{y\})$ *use y*
$(X_1, \{\})$ *End g*

Motivating example of value contexts method



Start main $(X_0, \{y, z\})$

C1 : Call f $(X_0, \{y, z\})$

R1 : Ret f $(X_0, \{y, z\})$

use z $(X_0, \{y, z\})$

C2 : Call g $(X_0, \{y\})$

R2 : Ret g $(X_0, \{\})$

End main $(X_0, \{\})$

Start f $(X_2, \{y\})(X_3, \{y, z\})$

End f $(X_2, \{y\})(X_3, \{y, z\})$

$(X_1, \{y\})$ *Start g*

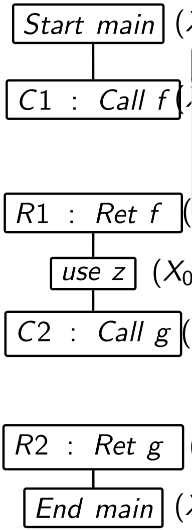
$(X_1, \{y\})$ *C3 : Call f*

$(X_1, \{y\})$ *R3 : Ret f*

$(X_1, \{y\})$ *use y*

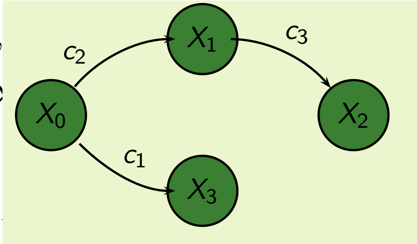
$(X_1, \{\})$ *End g*

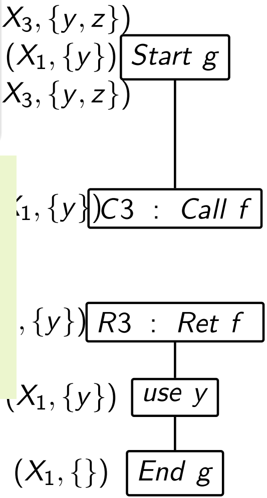
Motivating example of value contexts method



Context table

Context	Function	Entry	Exit
X_0	main	{}	{y,z}
X_1	g	{}	{y}
X_2	f	{y}	{y}
X_3	f	{y,z}	{y,z}





Part III

Our extensions to PRISM

Our extensions to PRISM

- Extended the kulang specification language to accept bidirectional data flow problems

Our extensions to PRISM

- Extended the kulang specification language to accept bidirectional data flow problems
- Extended the core solver to efficiently solve a bidirectional data flow problem.

Extensions to kulang

We extended the specifications so that they can accept

- Forward and backward flow functions and meet functions

Extensions to kulang

We extended the specifications so that they can accept

- Forward and backward flow functions and meet functions
- Forward and backward lattice types

Extensions to kulang

We extended the specifications so that they can accept

- Forward and backward flow functions and meet functions
- Forward and backward lattice types
- Forward and backward boundary values and top values

Role of the solver

Role of the solver :

- From the specifications, a java implementation of flow and meet functions are generated

Role of the solver

Role of the solver :

- From the specifications, a java implementation of flow and meet functions are generated
- The solver is a driver which uses the generated flow and meet functions to solve the data flow problem.

Role of the solver

Role of the solver :

- From the specifications, a java implementation of flow and meet functions are generated
- The solver is a driver which uses the generated flow and meet functions to solve the data flow problem.
- The solver maintains work list, context information and the intermediate results.

Extensions to solver

We extended the solver so that:

- The solver makes use of the code generated from bidirectional specifications

Extensions to solver

We extended the solver so that:

- The solver makes use of the code generated from bidirectional specifications
- The solver solves the bidirectional problem in the context sensitive manner

Extensions to solver

We extended the solver so that:

- The solver makes use of the code generated from bidirectional specifications
- The solver solves the bidirectional problem in the context sensitive manner
- The solver maintains the results of the analysis for different contexts.

Example of Bidirectional specifications

livenesslattice = set NamedEntity
pointstolattice = set tuple(NamedEntity,NamedEntity)

A ForwardMeet B = Meet(A,B)
A BackwardMeet B = A + B

BackwardFlowFunction(n : Unary, S : livenesslattice , T : pointstolattice)
where S is liveness at the OUT and T is pointsto at the IN

BackardFlowFunction(n : Binary, S : livenesslattice , T : pointstolattice)

Part IV

Building a scalable solver

Major factors affecting the running time of a solver

- Complexity of input program
- Complexity of flow functions
- Amount of data flow information maintained
- Path followed to achieve the fixed point

Complexity of input program

- Identify statements where gen will be constant
- Example where gen will not be constant

  p = **q

- Example where gen will be constant

  p = q

Complexity of flow functions

- Optimizations that can be handled by java compiler

  int a = c + b; // This assignment is dead if a is not used further

- Optimizations that cannot be handled by java compiler

  (a - a∩b) ∪ (b - a∩b)

  code produced by kulang translator :

  ```
  t1 = a.intersection(b)
  t2 = a.diff(t1)
  t3 = a.intersection(b)
  t4 = b.diff(t3)
  t5 = t2.union(t4)
  ```

  a ∪ b - a∩b (Simplified expression for the above expression)

Amount of data flow information maintained

- Separating accessible and non accessible information
- Bypassing global variables
- Techniques specific to data flow analysis

Amount of data flow information maintained

- Separating accessible and non accessible information

  A variable is accessible inside a function if :

  - It is local to the function
  - It is global
  - It is accessible through a pointer which is accessible inside the function

  $\{$p$-$ $>$a$\}$
  f() //No information will be passed to f
  $\{$p,a,b$\}$

  $\{$p$-$ $>$a$\}$
  f(p)$\{p, a\}$ will be passed to f
  $\{$p,a,b$\}$

Amount of data flow information maintained

- Bypassing global variables
  - Let G be set of global variables in the program
  - For every function, compute G1 which is set of global variables accessed from that function
  - At the function call, transfer G1 in the function and copy G-G1 directly across the function call

$\{G1 \cup (G - G1)\}$
f() // transfer G1 in the called function
$\{G2 \cup (G - G1)\}$ // G2 is at exit of the called function

Amount of data flow information maintained

```
int a,b,c,d

f()
{
use a
}



g()
{
f()
use b
}
```

access set of f = $\{a\}$
access set of g = $\{a, b\}$

Amount of data flow information maintained

- Techniques specific to data flow analysis
  - Do not explicitly maintain information of undef pointers

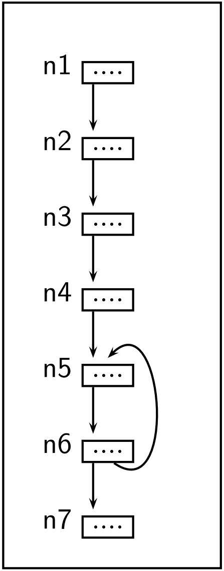
Do not explicitly maintain information of undef pointers

$x = \&b$
b $=$ 1;
c $=$ 2;
if(c$==$1)
$y = \&c$;
$//$ Insert $y->$ *undef* here
z $=$ x

Path followed to achieve the fixed point

- Solve the cfg in postorder in a backward analysis and preorder in a forward analysis
- Solve inner function calls first.

Path followed to achieve the fixedpoint



A possible processing order for a FIFO work list :

$n7, n6, n5, n4, n6, n3, n5, n2, n4, n1, n3, n2, n1$

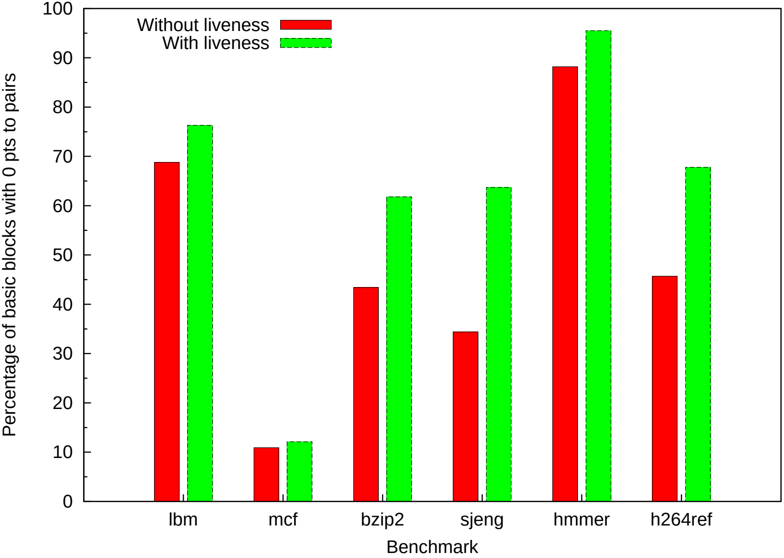
Optimal processing order :

$n7, n6, n5, n6, n5, n4, n3, n2, n1$

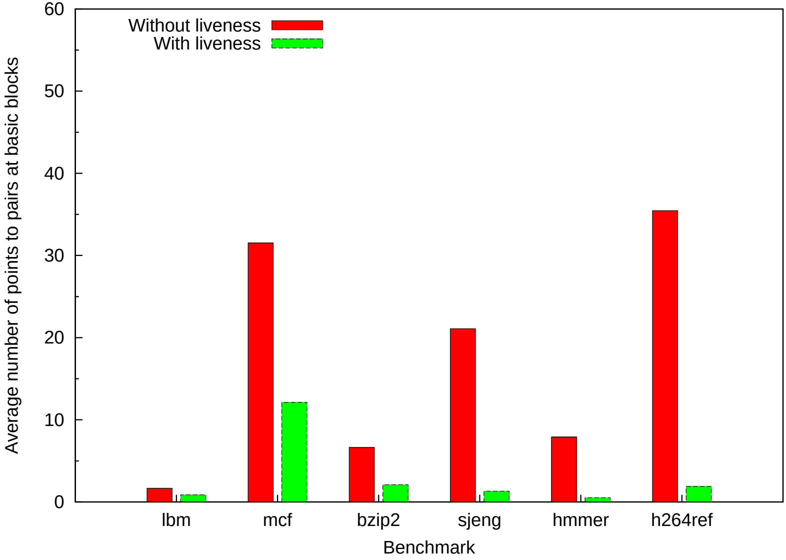
Part V

Performance measurements

Percentage of basic blocks having 0 pointsto pairs



Average pointsto pairs per basic blocks



Separating accessible and non accessible information vs non separation

Program	Without separating accessible and non accessible variables		Separating accessible and non accessible variables	
	Avg DF values	Execution time (ms)	Avg DF values	Execution time (ms)
mcf	7.46	488	2.51	312
lbm	10.01	434	0.58	258
bzip2	14.56	6815	1.2	1699
sjeng	-	-	0.62	6050
hmmer	-	-	0.4	3358
h264ref	-	-	1.01	33046

Table: Separating accessible and non accessible information vs non separation

Bypassing global variables

Program	Bypassing global variables			Not bypassing
	Execution time	Total global variables	Avg number of global variables accessed per context	Execution time
mcf	4280	26	5.86	4703
lbm	1639	4	0.7	1788
bzip2	21784	38	3.6	51445
sjeng	27116	25	1.85	92415
hmmer	27900	232	6.05	31268
h264ref	311021	464	43.95	1023911
cfe	4038702	1449	62.20	-

Measuring Scalability : Major operations

Operation	Percentage of time (<20kloc)	Percentage of time (>20kloc)
Flow functions	59.35 %	72.11 %
Meet function	4.97 %	3.32 %
CTD add/retrive	12.18%	9.43 %
Worklist add/retrive	10.12%	7.81 %
Store/load data flow information	5.24 %	3.27 %
Others	8.14 %	4.06 %

Table: Percentage of time required for individual operations

Measuring Scalability

1. The flow functions are a bottleneck as size of the program grows
2. The amount of time taken by flow function depends upon the size of set of data flow values processed by the flow function

Proposed method to remove the bottleneck

1. Implement globals bypassing inside a function.
2. Identify accessible information more precisely

Implement globals bypassing inside a function

```
f()
{
use x
}
```

$\{x, y, z\}$

```
g()
{
use y
f()
}
```

Identify accessible information more precisely

```
f(int** x)
{
printf(x)
}
```

```
main()
{
```

$\{x- > y, y- > z, z- > a\}$

```
f(x)
```

Only x is actually used inside f

```
}
```

Part VI

Future work

Future work

1. The current specifications allow specification of one forward and one backward analysis
2. Current specifications do not support specification of lattices of data flow problems. Hence meet function cannot be inferred but explicitly defined.

Future work

```
DATAFLOW_VAR dv1,dv2


dv1.lattice = powerset();
dv1.top = {}
dv1(n)_IN
{
< Stmttype >: {result =< expression >}
}
dv1(n)_OUT
{
< Stmttype >: {result =< expression >}
}
```

Part VII

Thank You !